Lecture 5: Image Classification with CNNs
Administrative

**Assignment 1** due Friday April 21, 11:59pm
- Important: tag your solutions with the corresponding hw question in gradescope!

**Assignment 2** will also be released on April 21
Administrative

Project proposal due **Monday Apr 24, 11:59pm**

Initial TA mentor: Canvas -> our course -> People -> Groups

Final TA mentor: assigned based on topic after proposal

Section on Friday will discuss final project guidelines
Thank you to everyone who participated in the high-resolution feedback in Week 2. The teaching team take your feedback seriously. The feedback you provided are crucial for us to continue improving the course.
Recap: Image Classification with Linear Classifier

\[
f(x, W) = Wx + b
\]

**Algebraic Viewpoint**

\[
f(x, W) = Wx
\]

**Visual Viewpoint**

One template per class

**Geometric Viewpoint**

Hyperplanes cutting up space
Recap: Loss Function

- We have some dataset of (x,y)
- We have a **score function**: \( s = f(x; W) = Wx \)
- We have a **loss function**:

\[
L_i = - \log\left( \frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \right) \quad \text{Softmax}
\]

\[
L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \quad \text{SVM}
\]

\[
L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W) \quad \text{Full loss}
\]
Recap: Optimization

- SGD
- SGD+Momentum
- RMSProp
- Adam
Problem: Linear Classifiers are not very powerful

Visual Viewpoint

Linear classifiers learn one template per class

Geometric Viewpoint

Linear classifiers can only draw linear decision boundaries
Last time: Neural Networks

Linear score function:

2-layer Neural Network

\[
f = Wx \\
f = W_2 \max(0, W_1 x)
\]
Last time: Computation Graph

\[ f = WX \]

\[ L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \]
Last time: Backpropagation

\[
\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x}
\]

“Downstream gradients”

\[
\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial y}
\]

“Upstream gradient”

“local gradient”
Backprop with Vectors

"Downstream gradients"

\[
\frac{\partial L}{\partial x} = \frac{\partial z}{\partial x} \frac{\partial L}{\partial z}
\]

Matrix-vector multiply

\[
[D_x \times D_z]
\]

\[
[D_y \times D_z]
\]

Jacobian matrices

"Local gradients"

\[
[D_x \times D_z]
\]

"Upstream gradient"

For each element of z, how much does it influence L?

Loss L still a scalar!

Fei-Fei Li, Yunzhu Li, Ruohan Gao
Backprop with Matrices (or Tensors)

\[ \frac{\partial L}{\partial x} = \frac{\partial z}{\partial x} \frac{\partial L}{\partial z} \]

\[ \frac{\partial L}{\partial y} = \frac{\partial z}{\partial y} \frac{\partial L}{\partial z} \]

“local gradients”

\[ (D_x \times M_x) \times (D_z \times M_z) \]

Jacobian matrices

Loss L still a scalar!

dL/dx always has the same shape as x!

“Downstream gradients”

Matrix-vector multiply

“Upstream gradient”

For each element of y, how much does it influence each element of z?

For each element of z, how much does it influence L?
CS231n: Deep Learning for Computer Vision

- Deep Learning Basics (Lecture 2 – 4)
- Perceiving and Understanding the Visual World (Lecture 5 – 12)
- Generative and Interactive Visual Intelligence (Lecture 13 – 16)
- Human-Centered Applications and Implications (Lecture 17 – 18)
Image Classification: A core task in Computer Vision

(assume given a set of labels)
{dog, cat, truck, plane, ...}

This image by Nikita is licensed under CC-BY 2.0
Pixel space

\[ f(x) = Wx \]
Image features

\[ f(x) = Wx \]

Feature Representation

Class scores
Example: Color Histogram
Example: Histogram of Oriented Gradients (HoG)

Divide image into 8x8 pixel regions
Within each region quantize edge direction into 9 bins

Example: 320x240 image gets divided into 40x30 bins; in each bin there are 9 numbers so feature vector has $30 \times 40 \times 9 = 10,800$ numbers

Lowe, "Object recognition from local scale-invariant features", ICCV 1999
Dalal and Triggs, "Histograms of oriented gradients for human detection," CVPR 2005
Example: Bag of Words

**Step 1: Build codebook**
- Extract random patches
- Cluster patches to form “codebook” of “visual words”

**Step 2: Encode images**

Fei-Fei and Perona, “A bayesian hierarchical model for learning natural scene categories”, CVPR 2005
Image Features
Image features vs. ConvNets

Feature Extraction

10 numbers giving scores for classes

training

Last Time: Neural Networks

Linear score function:

2-layer Neural Network

\[ f = W x \]

\[ f = W_2 \max(0, W_1 x) \]

The spatial structure of images is destroyed!
Next: Convolutional Neural Networks

Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1
Fei-Fei Li, Yunzhu Li, Ruohan Gao

A bit of history...

The **Mark I Perceptron** machine was the first implementation of the perceptron algorithm.

The machine was connected to a camera that used 20×20 cadmium sulfide photocells to produce a 400-pixel image.

The update rule:

\[
    w_{i}(t + 1) = w_{i}(t) + \alpha(d_{j} - y_{j}(t))x_{j,i},
\]

Frank Rosenblatt, ~1957: Perceptron
A bit of history...

Widrow and Hoff, ~1960: Adaline/Madaline

These figures are reproduced from Widrow 1960, Stanford Electronics Laboratories Technical Report with permission from Stanford University Special Collections.
A bit of history...

\[
\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial w_{ji}}
\]

Rumelhart et al., 1986: First time back-propagation became popular
A bit of history...

[Fei-Fei Li, Yunzhu Li, Ruohan Gao]

April 18, 2023

Lecture 5 - 28

Reinvigorated research in Deep Learning

Illustration of Hinton and Salakhutdinov 2006 by Lane McIntosh, copyright CS231n 2017
First strong results

**Acoustic Modeling using Deep Belief Networks**
Abdel-rahman Mohamed, George Dahl, Geoffrey Hinton, 2010

**Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition**
George Dahl, Dong Yu, Li Deng, Alex Acero, 2012

**Imagenet classification with deep convolutional neural networks**

A bit of history:

**Hubel & Wiesel, 1959**
RECEPTIVE FIELDS OF SINGLE NEURONES IN THE CAT'S STRIATE CORTEX

**1962**
RECEPTIVE FIELDS, BINOCULAR INTERACTION AND FUNCTIONAL ARCHITECTURE IN THE CAT'S VISUAL CORTEX

1968...
A bit of history

Topographical mapping in the cortex: nearby cells in cortex represent nearby regions in the visual field.
Hierarchical organization

Simple cells: Response to light orientation

Complex cells: Response to light orientation and movement

Hypercomplex cells: response to movement with an end point

Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n 2017
A bit of history:

Neocognitron

[Fukushima 1980]

“sandwich” architecture (SCSCSC…)
simple cells: modifiable parameters
complex cells: perform pooling
A bit of history:
Gradient-based learning applied to document recognition
[LeCun, Bottou, Bengio, Haffner 1998]
A bit of history:

ImageNet Classification with Deep Convolutional Neural Networks

[Krizhevsky, Sutskever, Hinton, 2012]

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

“AlexNet”
Fast-forward to today: ConvNets are everywhere

Classification

Retrieval

Fast-forward to today: ConvNets are everywhere

**Detection**

- [Faster R-CNN: Ren, He, Girshick, Sun 2015]

**Segmentation**

- [Farabet et al., 2012]

Figures copyright Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, 2015. Reproduced with permission.

Fast-forward to today: ConvNets are everywhere

NVIDIA Tesla line
(these are the GPUs on rye01.stanford.edu)

Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.
Fast-forward to today: ConvNets are everywhere

[Figures copyright Simonyan et al., 2014. Reproduced with permission.]

Illustration by Lane Mcintosh, photos of Katie Cumnock used with permission.

[Simonyan et al. 2014]
Fast-forward to today: ConvNets are everywhere

[Toshev, Szegedy 2014]

Fast-forward to today: ConvNets are everywhere

Figure copyright Levy et al. 2016. Reproduced with permission.

From left to right: public domain by NASA, usage permitted by ESA/Hubble, public domain by NASA, and public domain.

photos by Lane McIntosh. Copyright CS231n 2017.
**Whale recognition, Kaggle Challenge**

**Mnih and Hinton, 2010**
No errors

A white teddy bear sitting in the grass

A man in a baseball uniform throwing a ball

A woman is holding a cat in her hand

Minor errors

A man riding a wave on top of a surfboard

A cat sitting on a suitcase on the floor

Somewhat related

A woman standing on a beach holding a surfboard

All images are CC0 Public domain:

Captions generated by Justin Johnson using Neuraltalk2

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]
Convolutional Neural Networks
Recap: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

- **Input**: 3072 dimensions
- **Weights**: 10 x 3072
- **Activation**: 1 number

The result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)
Convolution Layer

32x32x3 image -> preserve spatial structure
Convolution Layer

32x32x3 image

5x5x3 filter

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”
Convolution Layer

32x32x3 image

5x5x3 filter

Filters always extend the full depth of the input volume

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”
Convolution Layer

A 32x32x3 image

A 5x5x3 filter \( w \)

1 number: the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. \( 5 \times 5 \times 3 = 75 \)-dimensional dot product + bias)

\[ w^T x + b \]
Convolution Layer
Convolution Layer
Convolution Layer
Convolution Layer
Convolution Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map
Convolution Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

consider a second, green filter

activation maps

Fei-Fei Li, Yunzhu Li, Ruohan Gao
April 18, 2023
Consider 6 filters, each 3x5x5

Stack activations to get a 6x28x28 output image!
Convolution Layer

3x32x32 image

Also 6-dim bias vector:

6 activation maps, each 1x28x28

Stack activations to get a 6x28x28 output image!
Convolution Layer

3x32x32 image

Also 6-dim bias vector:

Convolution Layer

6x3x5x5 filters

Stack activations to get a 6x28x28 output image!

28x28 grid, at each point a 6-dim vector

Slide inspiration: Justin Johnson
Convolution Layer

2x3x32x32 Batch of images

Also 6-dim bias vector:

6x3x5x5 filters

Convolution Layer

2x6x28x28 Batch of outputs

Slide inspiration: Justin Johnson
Convolution Layer

Batch of images $N \times C_{in} \times H \times W$

Convolution Layer

Also $C_{out}$-dim bias vector:

Batch of outputs $N \times C_{out} \times H' \times W'$

Slide inspiration: Justin Johnson
Preview: ConvNet is a sequence of Convolution Layers

CONV

e.g. 6 5x5x3 filters
Preview: ConvNet is a sequence of Convolution Layers

- CONV with 6 5x5x3 filters
- CONV with 10 5x5x6 filters
- ...
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions.

- Convolution layer: e.g., 6 \(5\times5\times3\) filters
- ReLU layer
- Next convolution layer: e.g., 10 \(5\times5\times6\) filters
- ReLU layer
- ...
Preview: What do convolutional filters learn?

Linear classifier: One template per class

Conv → ReLU
**Preview:** What do convolutional filters learn?

**MLP: Bank of whole-image templates**

```
32 32 28
```

- **Conv**
- **ReLU**

![MLP Diagram](image)
Preview: What do convolutional filters learn?

First-layer conv filters: local image templates
(Often learns oriented edges, opposing colors)

AlexNet: 64 filters, each 3x11x11
We call the layer convolutional because it is related to convolution of two signals:

\[ f[x, y] * g[x, y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1, n_2] \cdot g[x-n_1, y-n_2] \]

elementwise multiplication and sum of a filter and the signal (image)
A closer look at spatial dimensions:

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter

=> 5x5 output
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2 => 3x3 output!
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 3?
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
applied with stride 3?

doesn’t fit!
cannot apply 3x3 filter on 7x7 input with stride 3.
Output size:
\[(N - F) / \text{stride} + 1\]

e.g. \(N = 7, F = 3\):

\[
\begin{align*}
\text{stride 1} & \Rightarrow (7 - 3)/1 + 1 = 5 \\
\text{stride 2} & \Rightarrow (7 - 3)/2 + 1 = 3 \\
\text{stride 3} & \Rightarrow (7 - 3)/3 + 1 = 2.33
\end{align*}
\]
In practice: Common to zero pad the border

0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0

e.g. input 7x7
3x3 filter, applied with **stride 1**
**pad with 1 pixel** border => what is the output?

(recall:)
\[(N - F) / \text{stride} + 1\]
In practice: Common to zero pad the border

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

e.g. input 7x7
3x3 filter, applied with **stride 1**
**pad with 1 pixel** border => what is the output?

7x7 output!

(recall:)
\[(N + 2P - F) / \text{stride} + 1\]
In practice: Common to zero pad the border

e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with **1 pixel** border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with \((F-1)/2\). (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3
Remember back to…
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn’t work well.
Examples time:

Input volume: 32x32x3
10 5x5 filters with stride 1, pad 2

Output volume size: ?
Examples time:

Input volume: \(32 \times 32 \times 3\)
10 5x5 filters with stride 1, pad 2

Output volume size:
\[
\frac{32+2\times2-5}{1}+1 = 32 \text{ spatially, so } 32 \times 32 \times 10
\]
Examples time:

Input volume: \textbf{32x32x3}
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
Examples time:

Input volume: $32 \times 32 \times 3$
10 $5 \times 5$ filters with stride 1, pad 2

Number of parameters in this layer?
each filter has $5 \times 5 \times 3 + 1 = 76$ params (+1 for bias)
=> $76 \times 10 = 760$
Receptive Fields

For convolution with kernel size $K$, each element in the output depends on a $K \times K$ **receptive field** in the input.

Slide inspiration: Justin Johnson
Receptive Fields

Each successive convolution adds $K - 1$ to the receptive field size
With $L$ layers the receptive field size is $1 + L \times (K - 1)$

Be careful – “receptive field in the input” vs. “receptive field in the previous layer”

Slide inspiration: Justin Johnson
Each successive convolution adds $K - 1$ to the receptive field size. With $L$ layers, the receptive field size is $1 + L \times (K - 1)$.

Problem: For large images, we need many layers for each output to "see" the whole image.
Each successive convolution adds \( K - 1 \) to the receptive field size. With \( L \) layers, the receptive field size is \( 1 + L \times (K - 1) \).

**Problem:** For large images, we need many layers for each output to "see" the whole image image.

**Solution:** Downsample inside the network.
Solution: **Strided Convolution**

7x7 input (spatially) assume 3x3 filter applied **with stride 2**
Solution: Strided Convolution

7x7 input (spatially) assume 3x3 filter applied with stride 2

=> 3x3 output!
Convolution layer: summary

Let’s assume input is $W_1 \times H_1 \times C$
Conv layer needs 4 hyperparameters:
  - Number of filters $K$
  - The filter size $F$
  - The stride $S$
  - The zero padding $P$
This will produce an output of $W_2 \times H_2 \times K$
where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$
Number of parameters: $F^2CK$ and $K$ biases
Convolution layer: summary

Let’s assume input is $W_1 \times H_1 \times C$

Conv layer needs 4 hyperparameters:
- Number of filters $K$
- The filter size $F$
- The stride $S$
- The zero padding $P$

This will produce an output of $W_2 \times H_2 \times K$
where:
- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: $F^2CK$ and $K$ biases

Common settings:
- $K = (\text{powers of 2, e.g. 32, 64, 128, 512})$
  - $F = 3, S = 1, P = 1$
  - $F = 5, S = 1, P = 2$
  - $F = 5, S = 2, P = ?$ (whatever fits)
  - $F = 1, S = 1, P = 0$
(btw, 1x1 convolution layers make perfect sense)

1x1 CONV
with 32 filters

(each filter has size 1x1x64, and performs a 64-dimensional dot product)
(btw, 1x1 convolution layers make perfect sense)

1x1 CONV with 32 filters

(each filter has size 1x1x64, and performs a 64-dimensional dot product)
Conv layer needs 4 hyperparameters:
- Number of filters \( K \)
- The filter size \( F \)
- The stride \( S \)
- The zero padding \( P \)

Example: CONV layer in PyTorch

```python
Conv2d
```
Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size \((N, C_{in}, H, W)\) and output
\((N, C_{out}, H_{out}, W_{out})\) can be precisely described as:

\[
\text{out}(N, C_{out}) = \text{bias}(C_{out}) + \sum_{k=0}^{C_{in}-1} \text{weight}(C_{out}, k) \ast \text{input}(N, k)
\]

where \( \ast \) is the valid 2D cross-correlation operator, \( N \) is a batch size, \( C \) denotes a number of channels, \( H \) is a height of input planes in pixels, and \( W \) is width in pixels.

- \( \text{stride} \) controls the stride for the cross-correlation, a single number or a tuple.
- \( \text{padding} \) controls the amount of implicit zero-paddings on both sides for padding number of points for each dimension.
- \( \text{dilation} \) controls the spacing between the kernel points; also known as the à trous algorithm. It is harder to describe, but this link has a nice visualization of what dilation does.
- \( \text{groups} \) controls the connections between inputs and outputs. \text{in_channels} and \text{out_channels} must both be divisible by \text{groups}. For example,
  - At \text{groups}=1, all inputs are convolved to all outputs.
  - At \text{groups}=2, the operation becomes equivalent to having two conv layers side by side, each seeing half the input channels, and producing half the output channels, and both subsequently concatenated.
  - At \text{groups}>\text{in_channels}, each input channel is convolved with its own set of filters, of size: \( \left[ \frac{C_{out}}{C_{in}} \right] \).

The parameters \text{kernel_size}, \text{stride}, \text{padding}, \text{dilation} can either be:
- a single \text{int} - in which case the same value is used for the height and width dimension
- a \text{tuple} of two \text{ints} - in which case, the first \text{int} is used for the height dimension, and the second \text{int} for the width dimension

PyTorch is licensed under BSD 3-clause.
The brain/neuron view of CONV Layer

32x32x3 image
5x5x3 filter

1 number: the result of taking a dot product between the filter and this part of the image (i.e. 5*5*3 = 75-dimensional dot product)
The brain/neuron view of CONV Layer

32x32x3 image
5x5x3 filter

1 number:
the result of taking a dot product between
the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)
The brain/neuron view of CONV Layer

E.g. with 5 filters, CONV layer consists of neurons arranged in a 3D grid (28x28x5)

There will be 5 different neurons all looking at the same region in the input volume
Reminder: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

Each neuron looks at the full input volume

= \sum w_i x_i

1 number: the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)
Pooling layer
- makes the representations smaller and more manageable
- operates over each activation map independently

![Diagram of pooling layer](image)
MAX POOLING

Single depth slice

max pool with 2x2 filters and stride 2
MAX POOLING

Single depth slice

\[
\begin{array}{cccc}
1 & 1 & 2 & 4 \\
5 & 6 & 7 & 8 \\
3 & 2 & 1 & 0 \\
1 & 2 & 3 & 4 \\
\end{array}
\]

max pool with 2x2 filters and stride 2

\[
\begin{array}{cc}
6 & 8 \\
3 & 4 \\
\end{array}
\]

- No learnable parameters
- Introduces spatial invariance
Pooling layer: summary

Let’s assume input is $W_1 \times H_1 \times C$

Conv layer needs 2 hyperparameters:
- The spatial extent $F$
- The stride $S$

This will produce an output of $W_2 \times H_2 \times C$ where:
- $W_2 = (W_1 - F)/S + 1$
- $H_2 = (H_1 - F)/S + 1$

Number of parameters: 0
Fully Connected Layer (FC layer)
- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks
ConvNetJS CIFAR-10 demo

This demo trains a Convolutional Neural Network on the CIFAR-10 dataset in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used this python script to parse the original files (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and vertically.

By default, in this demo we’re using Adadelta which is one of per-parameter adaptive step size methods, so we don’t have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you’d like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to @karpathy.

http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html
Summary

- ConvNets stack CONV, POOL, FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Historically architectures looked like
  \[
  [(\text{CONV-RELU})*N-\text{POOL?}]*M-(\text{FC-RELU})*K,\text{SOFTMAX}
  \]
  where $N$ is usually up to $\sim 5$, $M$ is large, $0 \leq K \leq 2$.
- But recent advances such as ResNet/GoogLeNet have challenged this paradigm
Next time: CNN Architectures